Audio Processing and Playback

Session 316

These are confidential sessions—please refrain from streaming, blogging, or taking pictures.

AVFoundation

James McCartney
Core Audio Engineering

AVFoundation

- AVAudioPlayer
- AVAudioRecorder
- AVAudioSession

AVAudioPlayer

- Simplest way to play a sound
- Layered on top of AudioQueue, AudioFile
- Plays file types supported by AudioFile, including
  - .caf .m4a .mp3 .aif .wav .au .snd .aac
- Provides basic playback operations
  - Create, prepare, play, pause, skip, stop
- Supports volume, metering, looping
- Uses a delegate for notifications:
  - Finished playing, decode errors, interruptions

Creating an AVAudioPlayer

```
NSString* filename = [NSString stringWithUTF8String: utf8_string];
NSURL* url = [[NSURL alloc] initFileURLWithPath: filename];
[filename release];
AVAudioPlayer* player = [[AVAudioPlayer alloc] initWithContentsOfURL: url error: nil];
[url release];
```

```
NSData* data = ...;
AVAudioPlayer* player = [[AVAudioPlayer alloc] initWithData: data error: nil];
```

prepareToPlay

```
ok = [player prepareToPlay];
```

- Gets ready to play the sound
  - Creates an AudioQueue
  - Allocates buffers
  - Performs priming
- Returns YES if successful, NO if failed
- Happens automatically on play

play

```
ok = [player play];
```

- Starts playing the sound
- Resumes playing if paused or stopped

pause

```
[player pause];
```

- Playback is stopped
- Player remains prepared to play
  - AudioQueue and buffers are still allocated
- Call ‘play’ to resume playing from where it left off

stop

```
[player stop];
```

- Playback is stopped
- Player is no longer ‘prepared’
  - AudioQueue and buffers are disposed
- Call ‘play’ to resume playing from where it left off

stop

```
[player stop];
```

- Playback is stopped
- Player is no longer ‘prepared’
  - AudioQueue and buffers are disposed
- Call ‘play’ to resume playing from where it left off
  - ‘stop’ does not reset the play position to the start!
    - Works just like ‘stop’ on a tape deck
    - Set currentTime to zero to go back to the start

currentTime

```
NSTimeInterval secs = player.currentTime;

player.currentTime = secs + 1.0;
```

- currentTime is the current position in the file in seconds
- You can set currentTime at any time
- You can use setting currentTime to implement a ‘fast forward’ or reverse
  - See demo…

metering

```
player.meteringEnabled = YES;

        ...

[player updateMeters];
double avgPwrLeft = [player averagePowerForChannel: 0];
double avgPwrRight = [player averagePowerForChannel: 1];
```

- Enable metering if you want it
- Then periodically
  - Update the meter values
  - Get the values of the meters (in decibels)

Other Properties

- volume—(read/write)
- numberOfLoops—(read/write)
- numberOfChannels—(read only)
- duration—(read only)

Delegate Methods

```
player.delegate = myDelegate;
```

- When certain events happen, your delegate gets called
- The player finished playing

  - (void)audioPlayerDidFinishPlaying:(AVAudioPlayer *)player successfully:(BOOL)flag

- There was a decode error

  - (void)audioPlayerDecodeErrorDidOccur:(AVAudioPlayer *)player error:(NSError*)error

- An interruption began

  - (void)audioPlayerBeginInterruption:(AVAudioPlayer *)player

- An interruption ended

  - (void)audioPlayerEndInterruption:(AVAudioPlayer *)player

Resuming After an Interruption

```
- (void)audioPlayerEndInterruption:(AVAudioPlayer *)player
{
 if (iStillWantToPlay)
            [player play];
}
```

- Playback was stopped at begin interruption
- Call ‘play’ to resume playing

avTouch Demo: Implementing Fast Forward

```
#define SKIP_INTERVAL 0.1   // amount to play between skips
#define SKIP_TIME 0.3       // amount to skip on rewind or fast forward
```

avTouch Demo: Implementing Fast Forward

```
- (IBAction)ffwButtonPressed:(UIButton *)sender
{
   if (_ffwTimer)
     [_ffwTimer invalidate];

   _ffwTimer = [NSTimer scheduledTimerWithTimeInterval: SKIP_INTERVAL
                                                target: self
                                              selector: @selector(ffwd)
                                              userInfo: self._player
                                               repeats: YES];
}
```

avTouch Demo: Implementing Fast Forward

```
- (IBAction)ffwButtonReleased:(UIButton *)sender
{
    if (_ffwTimer)
        [_ffwTimer invalidate];

    _ffwTimer = nil;
}
```

avTouch Demo: Implementing Fast Forward

```
- (void)ffwd
{
   AVAudioPlayer *player = _ffwTimer.userInfo;
   player.currentTime += SKIP_TIME;
   [self updateCurrentTimeForPlayer:player];
}
```

Demo

AVAudioRecorder

- Records sound
- Interface very similar to AVAudioPlayer

Creating an AVAudioRecorder

```
NSString* filename = [NSString stringWithUTF8String: utf8_string];
NSURL* url = [[NSURL alloc] initFileURLWithPath: filename];
[filename release];
NSMutableDictionary* settings = [[NSMutableDictionary alloc]
                                        initWithObjects: values
                                                forKeys: keys
                                                  count: count];

AVAudioRecorder* recorder = [[AVAudioRecorder alloc] initWithURL: url
                                                        settings: settings
                                                           error: nil];

[url release];
[settings release];
```

Settings Dictionary

- A dictionary of key/value pairs that specify
  - What format to record
  - Number of channels
  - Sample rate
  - For LPCM
    - Bit depth, endian-ness
  - For encoded formats
    - Quality, bit rate, bit depth

Settings Dictionary Keys

Key	Value Type
AVFormatIDKey	integer
AVSampleRateKey	float
AVNumberOfChannelsKey	integer
AVLinearPCMBitDepthKey	integer
AVLinearPCMIsBigEndianKey	boolean
AVLinearPCMIsFloatKey	boolean
AVEncoderAudioQualityKey	integer
AVEncoderBitRateKey	integer
AVEncoderBitDepthKey	integer
AVSampleRateConverterQualityKey	integer

Settings Dictionary Example

```
id keys[5], values[5];
keys[0] = AVFormatIDKey;             values[0] = [[NSNumber alloc]   initWithInt: kAudioFormatMPEG4AAC ];
keys[1] = AVSampleRateKey;           values[1] = [[NSNumber alloc] initWithFloat: 44100. ];
keys[2] = AVNumberOfChannelsKey;     values[2] = [[NSNumber alloc]   initWithInt: 2 ];
keys[3] = AVEncoderBitRateKey;       values[3] = [[NSNumber alloc]   initWithInt: 128000 ];
keys[4] = AVEncoderAudioQualityKey; values[4] = [[NSNumber alloc]   initWithInt: AVAudioQualityMax ];
NSMutableDictionary* settings = [[NSMutableDictionary alloc]
                                          initWithObjects: values
                                                  forKeys: keys
                                                    count: 5];
for (int i = 0; i < 5; ++i) [values[i] release];
```

Recording

- These work like the corresponding AVAudioPlayer methods

```
ok = [recorder prepareToRecord];

ok = [recorder record];

[recorder pause];

[recorder stop];
```

Metering

```
recorder.meteringEnabled = YES;

        ...

[recorder updateMeters];
double avgPwrLeft = [recorder averagePowerForChannel: 0];
double avgPwrRight = [recorder averagePowerForChannel: 1];
```

- Works exactly the same as AVAudioPlayer

Delegate Methods

```
recorder.delegate = myDelegate;
```

- When certain events happen, your delegate gets called
- The recorder finished recording

```
- (void)audioRecorderDidFinishRecording:(AVAudioRecorder *)recorder
                                successfully:(BOOL)flag
```

- There was an encode error

```
- (void)audioRecorderEncodeErrorDidOccur:(AVAudioRecorder *) recorder
                                    error:(NSError*)error
```

- An interruption began

```
- (void)audioRecorderBeginInterruption:(AVAudioRecorder *) recorder
```

- An interruption ended

```
- (void)audioRecorderEndInterruption:(AVAudioRecorder *) recorder
```

AVFoundation

- AVAudioPlayer
  - Simple playback
- AVAudioRecorder
  - Simple recording

OpenAL: Best Practices

Bob Aron
Core Audio Engineering

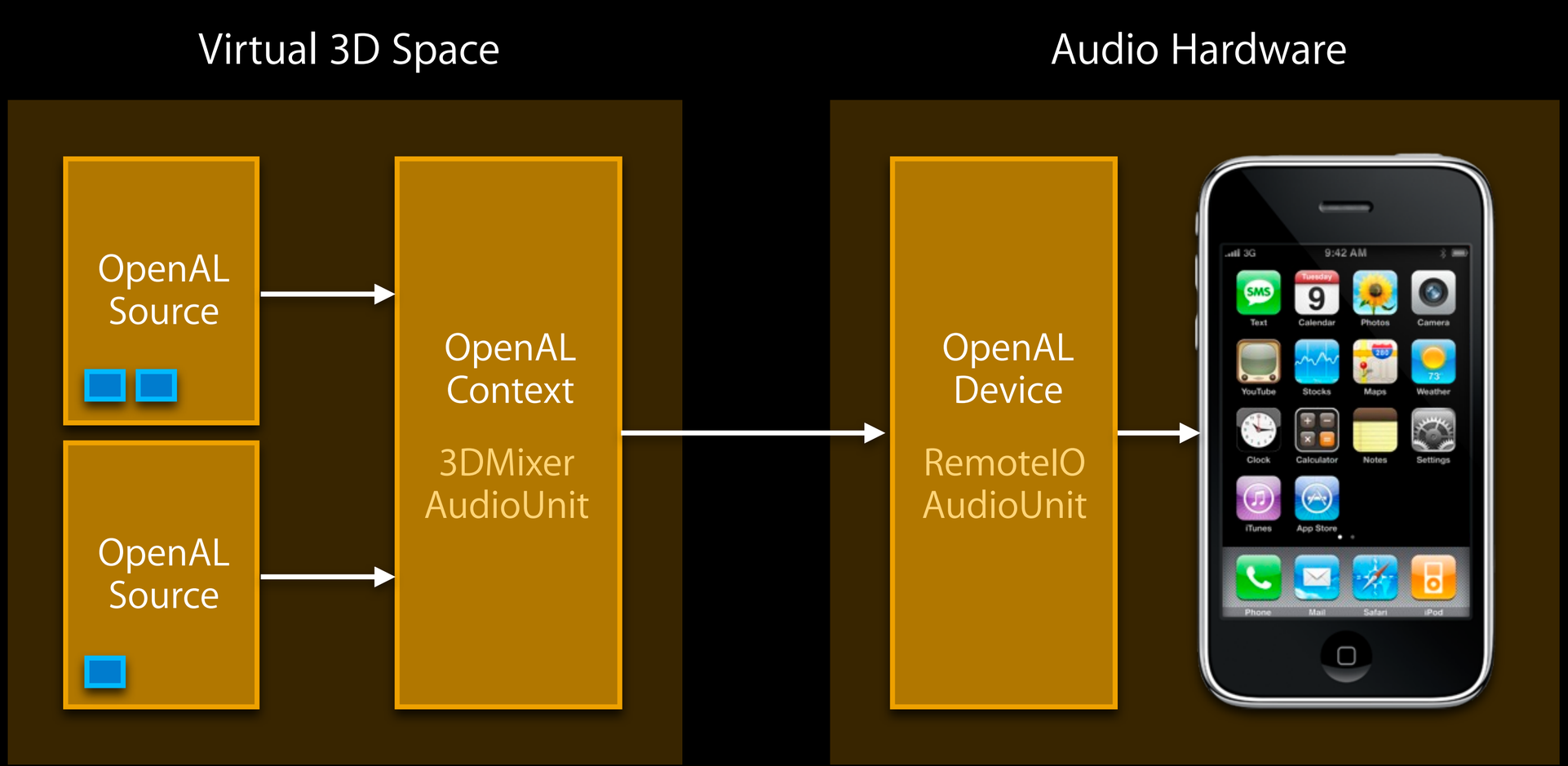
Introduction

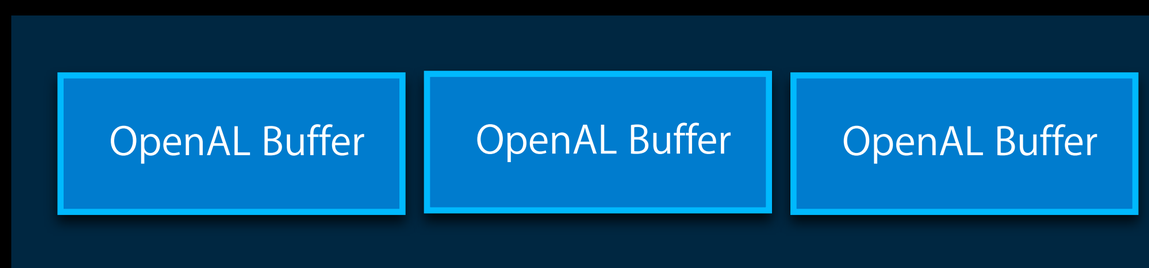
- OpenAL review
- Best practices
  - Efficient usage patterns
  - Usage patterns to avoid

OpenAL: Review

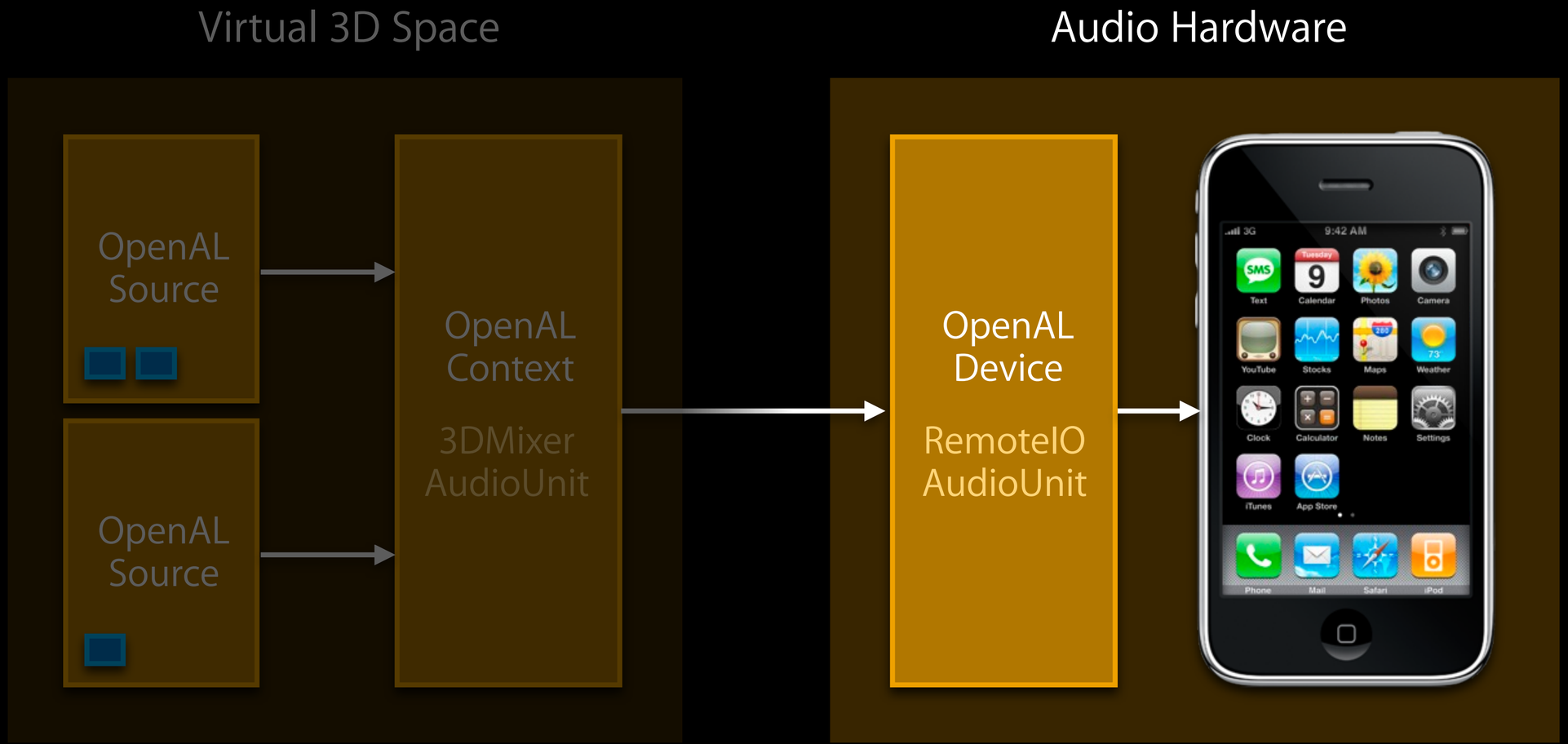
- What is OpenAL?
  - OpenGL-like library for Audio
  - Cross platform API
  - Game development
    - www.openal.org
- Apple's commitment to OpenAL
  - OpenAL.framework released on 10.4
  - OpenAL 1.1 implementations
    - OS X: 10.4.7 and greater
    - iPhone OS: 2.0 and greater

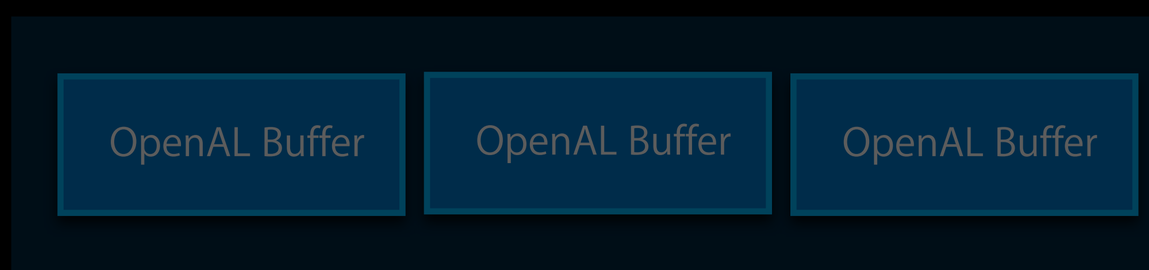
OpenAL: Review



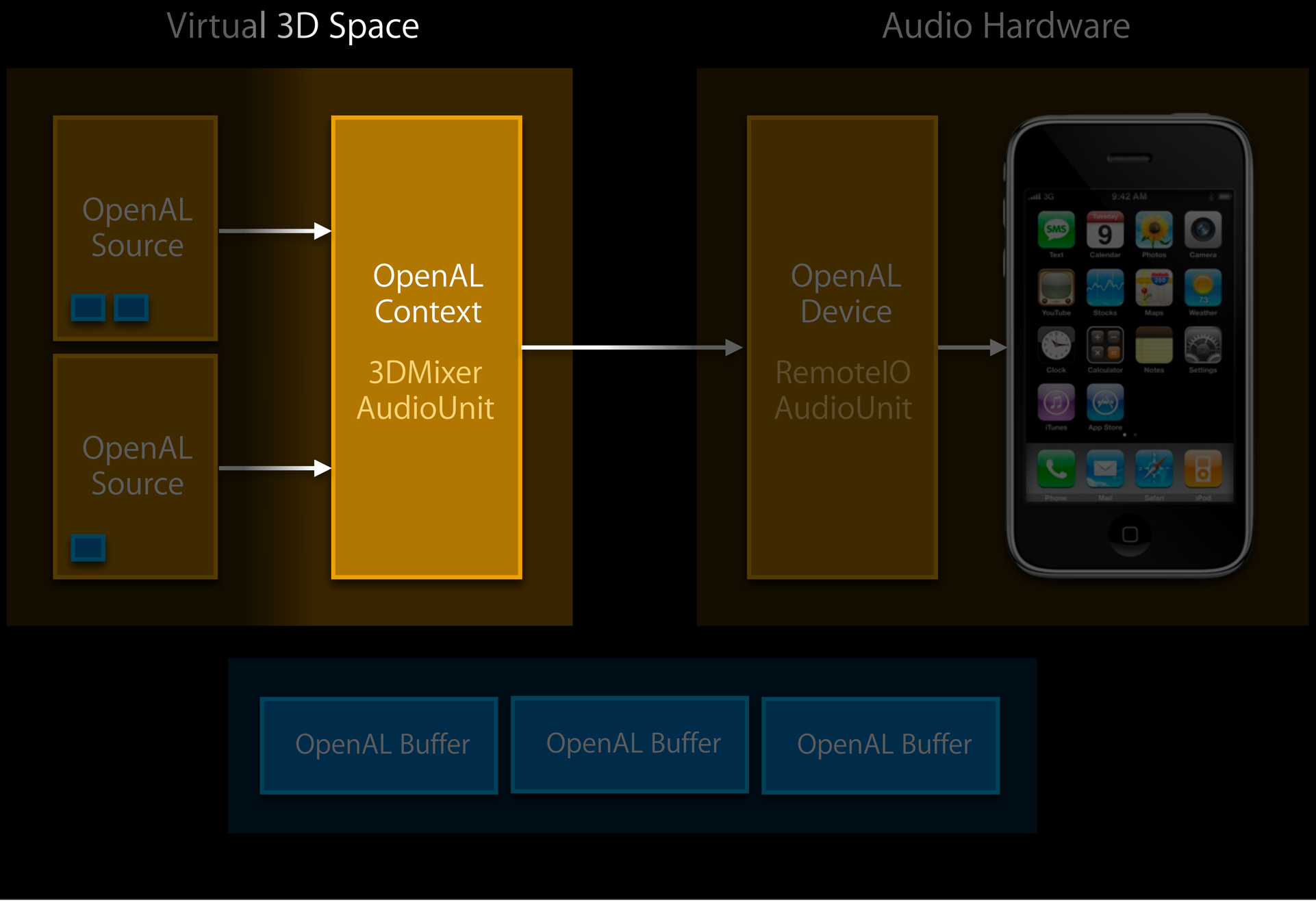


OpenAL: Review

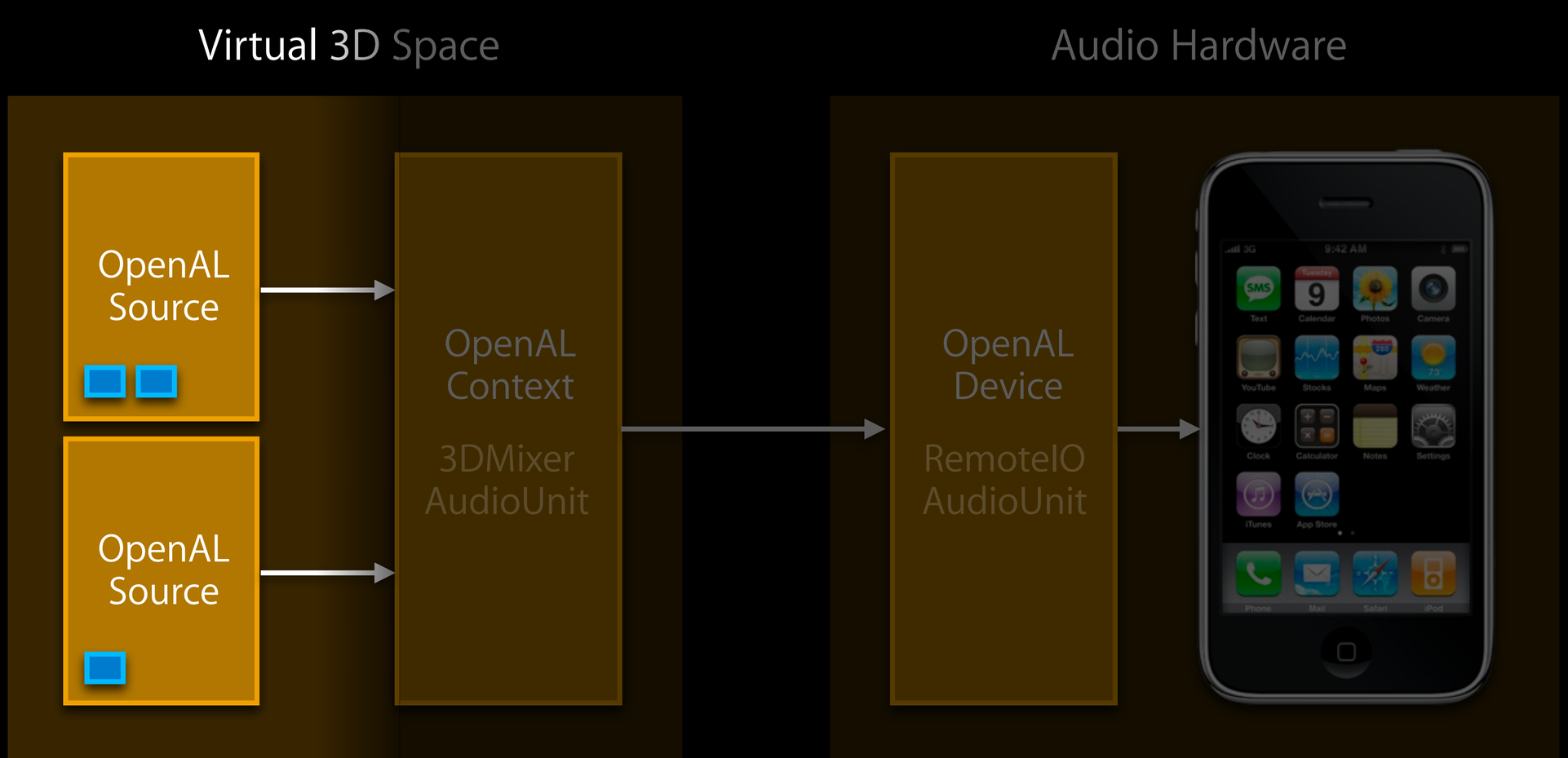




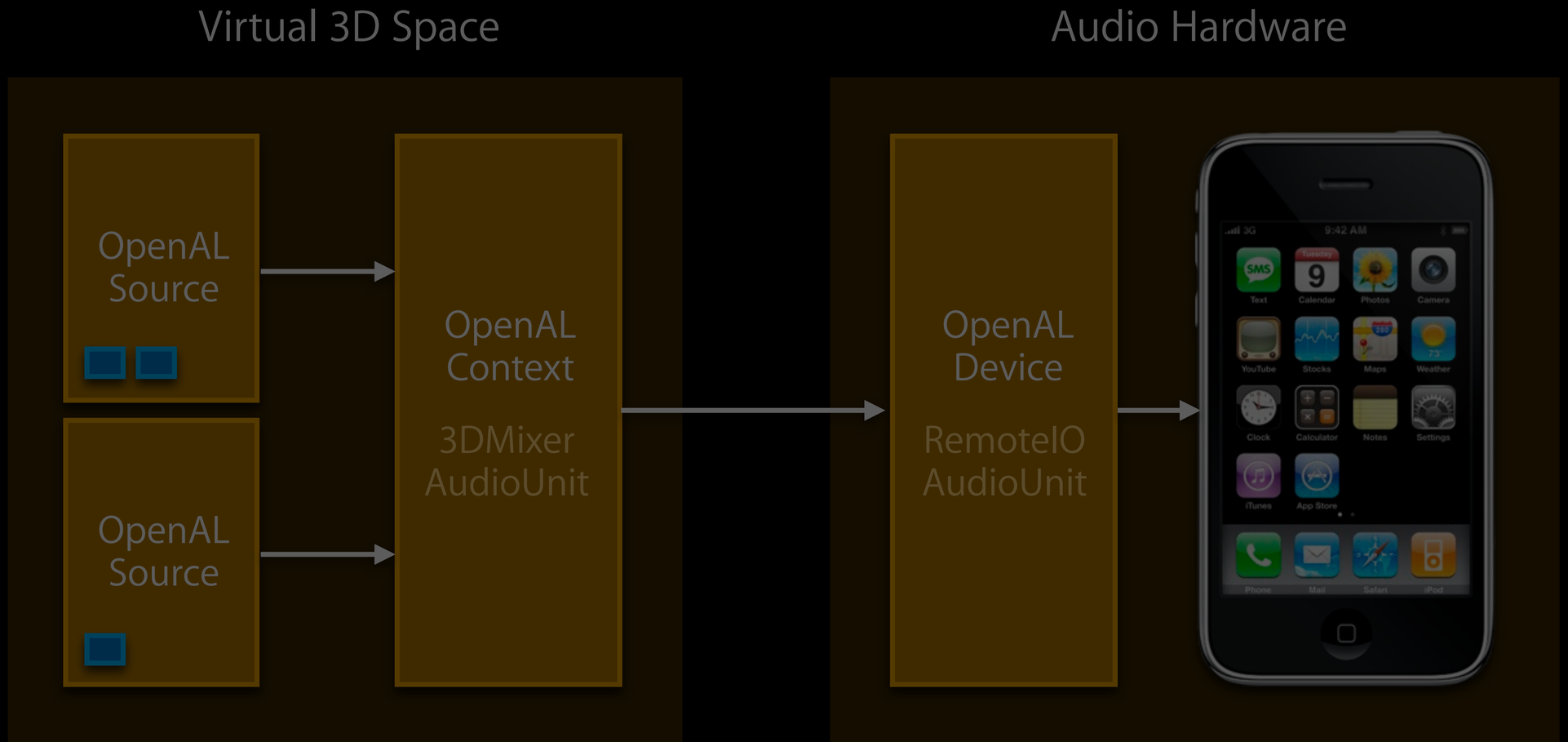
OpenAL: Review

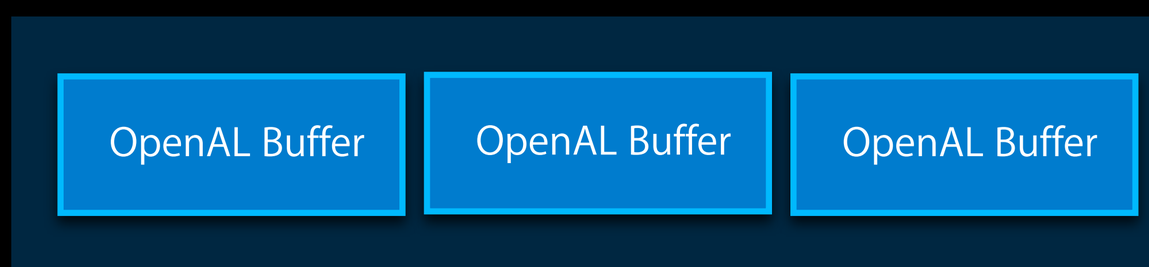


OpenAL: Review



OpenAL: Review





OpenAL Best Practice

Handling interruptions

- Existing applications (iPhone 2.x OS)—Destroy/reconstruct context

```
void interruptionListener(void* inClientData, UInt32 inInterruptionState)
{
    if (inInterruptionState == kAudioSessionBeginInterruption) {
        // Save State of OALContext and related OALSources
        StoreMyContextState(myContext);
        alcDestroyContext(myContext);
    }
    if (inInterruptionState == kAudioSessionEndInterruption) {
        ALCcontext*   myContext = alcCreateContext(myOpenALDevice, NULL);
        // Restore OALContext State and related OALSources
        RestoreMyContextState(myContext);
    }
}
```

OpenAL Best Practice

Handling interruptions

- iPhone 3.0 OS applications—Use alcMakeContextCurrent()

```
void interruptionListener(void* inClientData, UInt32 inInterruptionState)
{
     if (inInterruptionState == kAudioSessionBeginInterruption) {
        alcMakeContextCurrent (NULL);
     }
     else if (inInterruptionState == kAudioSessionEndInterruption) {
        alcMakeContextCurrent (myContext);
     }
}
```

OpenAL Best Practice

Handling interruptions

- iPhone 2.x OS compatible applications can decide at runtime

```
-(NSInteger) getMajorOSVersion {
  NSString *versionString = [[UIDevice currentDevice] systemVersion];
  return [[[versionString componentsSeparateByString:@”.”] objectAtIndex:0] intValue];
}
```

```
void interruptionListener(void* inClientData, UInt32 inInterruptionState)
{
     if ([self getMajorOSVersion] >= 3) {
          // 3.0 OS mechanism: Use alcMakeContextCurrent()
     }
     else {
          // 2.x OS mechanism: Use alcCreateContext()/alcDestroyContext()
     }
}
```

OpenAL Best Practice

Polling source object

- Playback state or buffers processed
- Polling too often may affect performance

```
ALint   numBuffersProcessed  = 0;
while (numBuffersProcessed < 1)
{
     alGetSourcei(mySourceID, AL_BUFFERS_PROCESSED, numBuffersProcessed);

}
```

OpenAL Best Practice

Polling source object

- Playback state or buffers processed
- Polling too often may affect performance

```
ALint   numBuffersProcessed  = 0;
while (numBuffersProcessed < 1)
{
    alGetSourcei(mySourceID, AL_BUFFERS_PROCESSED, numBuffersProcessed);
    usleep(10000);
}
```

OpenAL Best Practice

Reuse source objects

- Inefficient usage pattern commonly found

```
PlaySound (ALuint  myBufferID)
{
    // Create a Source
    ALuint mySourceID;
    alGenSources(1, &mySourceID);

    // Attach Audio Data to Source for Playback
    alSourcei(mySourceID, AL_BUFFER, myBufferID);

    // Play Sound
    alSourcePlay(mySourceID);

    // wait until sound finishes playing
    ...

    // Delete Source object
    alDeleteSources(1, &mySourceID);
}
```

OpenAL Best Practice

Reuse source objects

- When finished rendering, release buffers from source

```
PlaySound (ALuint  myBufferID, ALuint  mySoundID)
{
    // Attach Audio Data to Source for Playback
    alSourcei(mySourceID, AL_BUFFER, myBufferID);

    // Play Sound
    alSourcePlay(mySourceID);

    // wait until sound finishes playing
    ...

    // Detach Buffer from Source
    alSourcei(mySourceID, AL_BUFFER, AL_NONE);
}
```

OpenAL Buffer: Providing Data

- It's a data storage object
- Two ways to provide data
  - alBufferData() -
    - OpenAL copies data to internal buffers
    - Application can release data memory upon completion of alBufferData()

```
alBufferData (myBufferID, AL_FORMAT_MONO16, myAudioData, 4096, 44100);
```

  - alBufferDataStatic() -
    - OpenAL uses memory provided by application
    - Application must NOT dispose data memory while in use

```
alBufferDataStatic (myBufferID, AL_FORMAT_MONO16, myAudioData, 4096, 44100);
```

OpenAL Best Practice

Use alBufferDataStatic

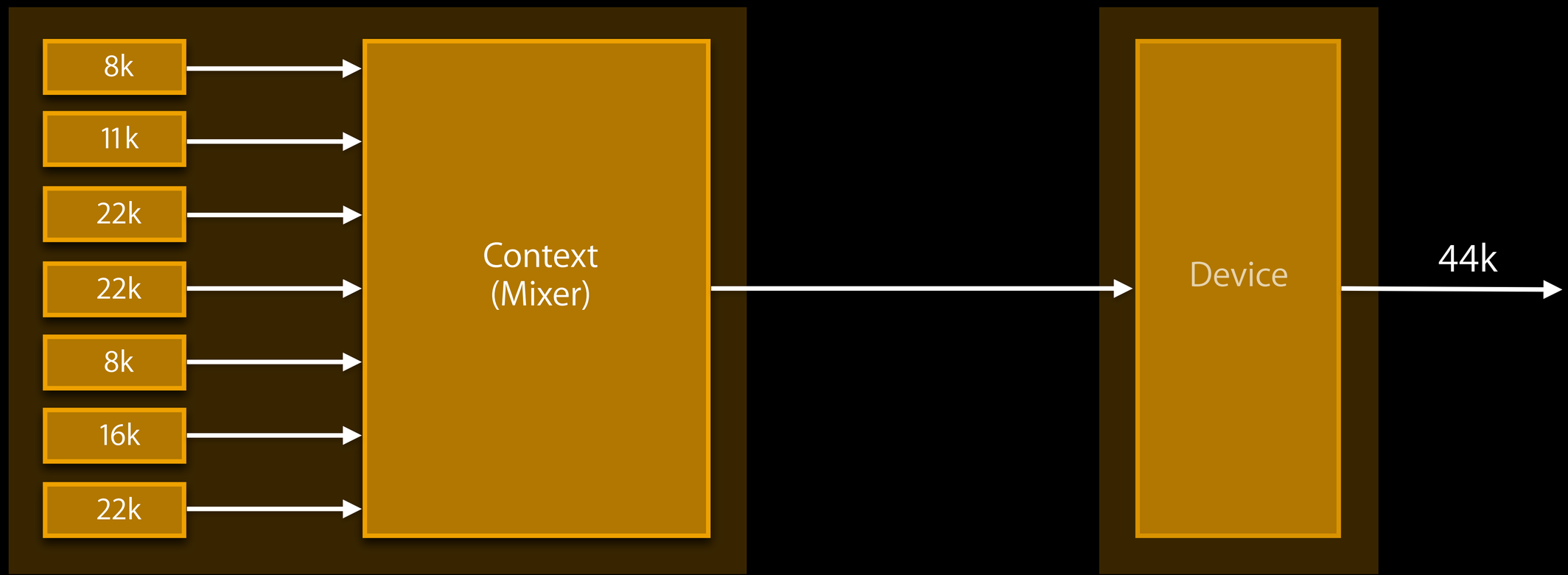
- Avoids memcpy()
- Application must NOT release audio data memory until successfully calling
  - alDeleteBuffers()
  - alBufferData()
  - alBufferDataStatic()
- Error checking is critical before releasing data memory

```
alDeleteBuffers (1, myBufferID);
if (alGetError() == AL_NO_ERROR)
    DisposeMyBufferData(myBufferAudioData);
```

OpenAL Best Practice

Tuning your assets

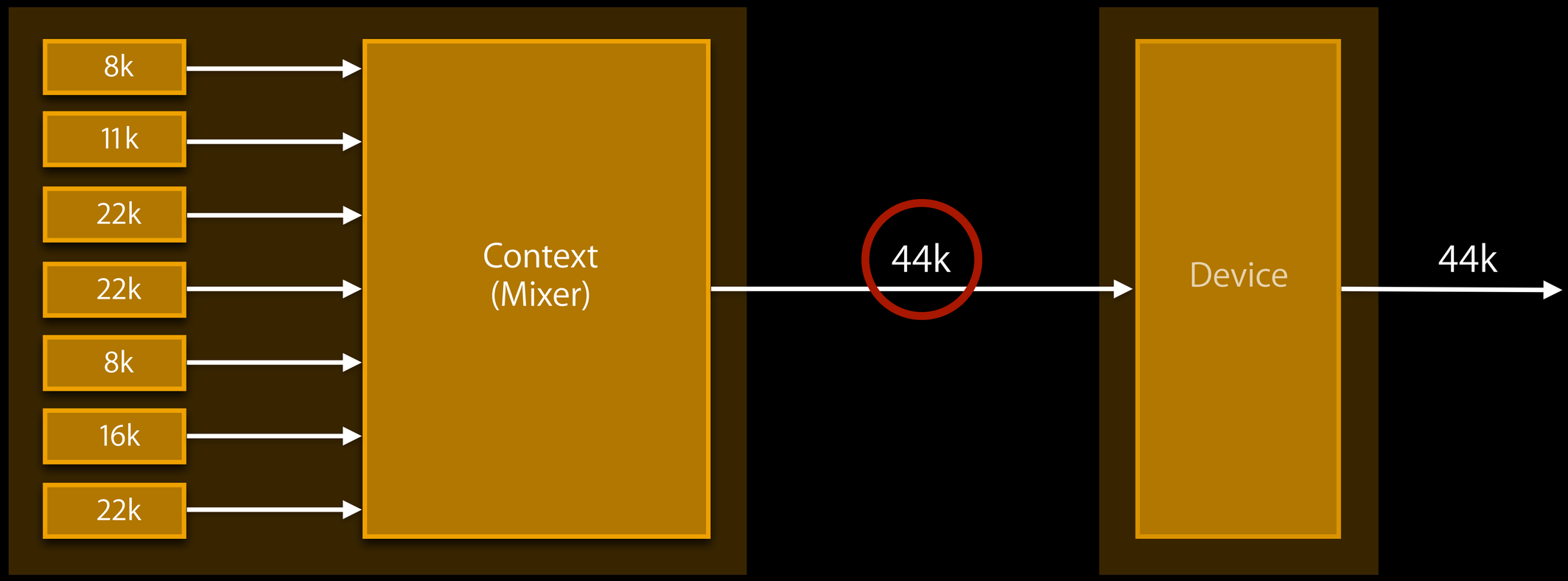
- It’s all about the sample rate
  - Audio data, mixer and device sample rates



OpenAL Best Practice

Tuning your assets

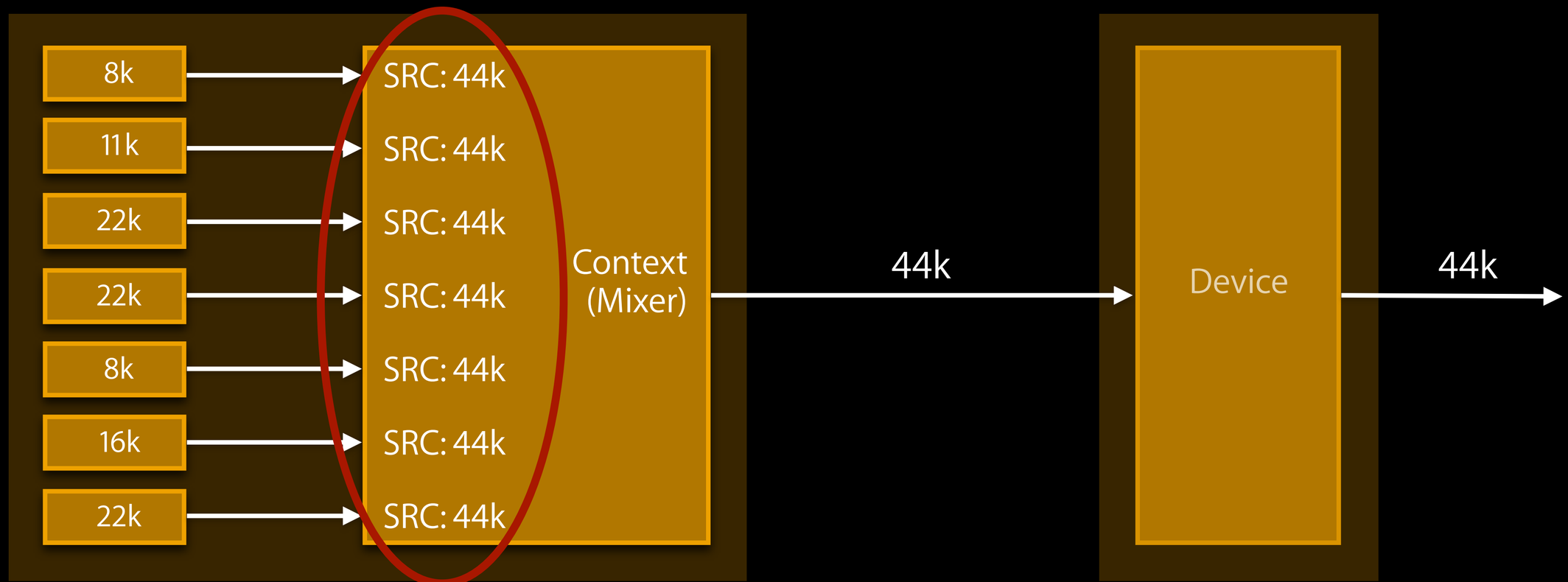
- By default, the mixer renders at device HW rate: 44k



OpenAL Best Practice

Tuning your assets

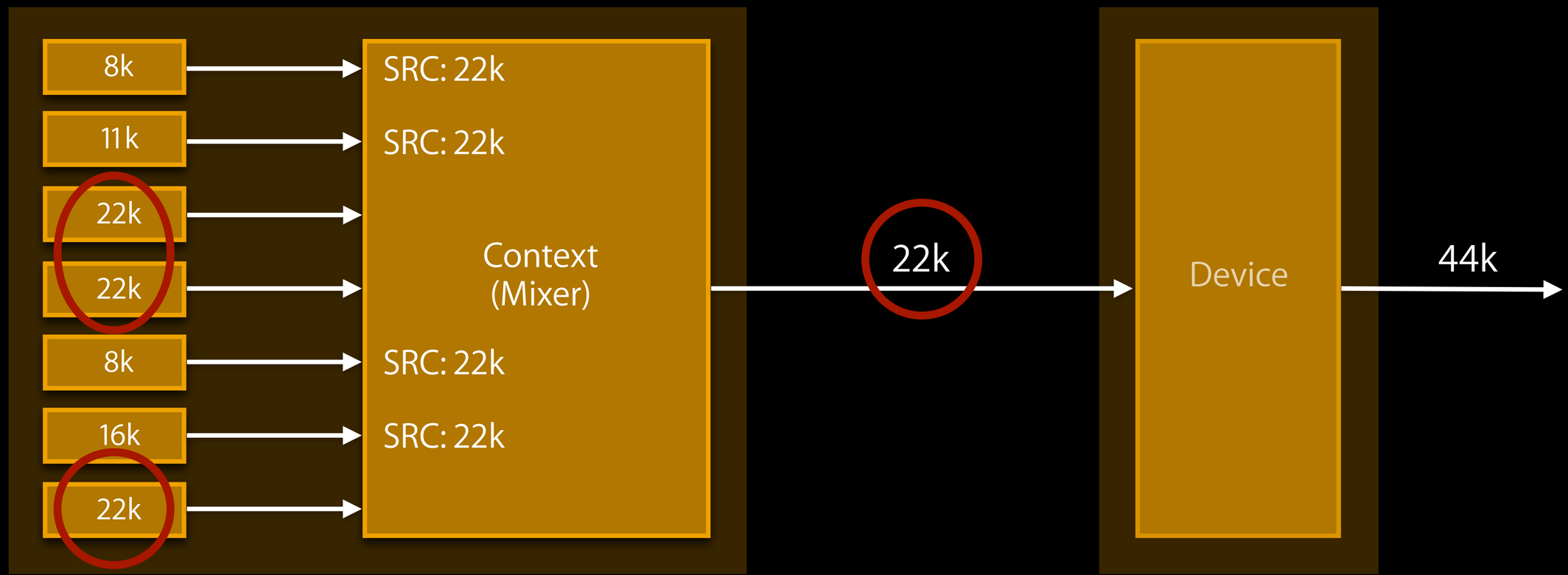
- By default, the mixer renders at device HW rate: 44k
  - More processing: Sample rate conversion required
  - More data to mix: 44,100 samples per second for each source



OpenAL Best Practice

Tuning your assets

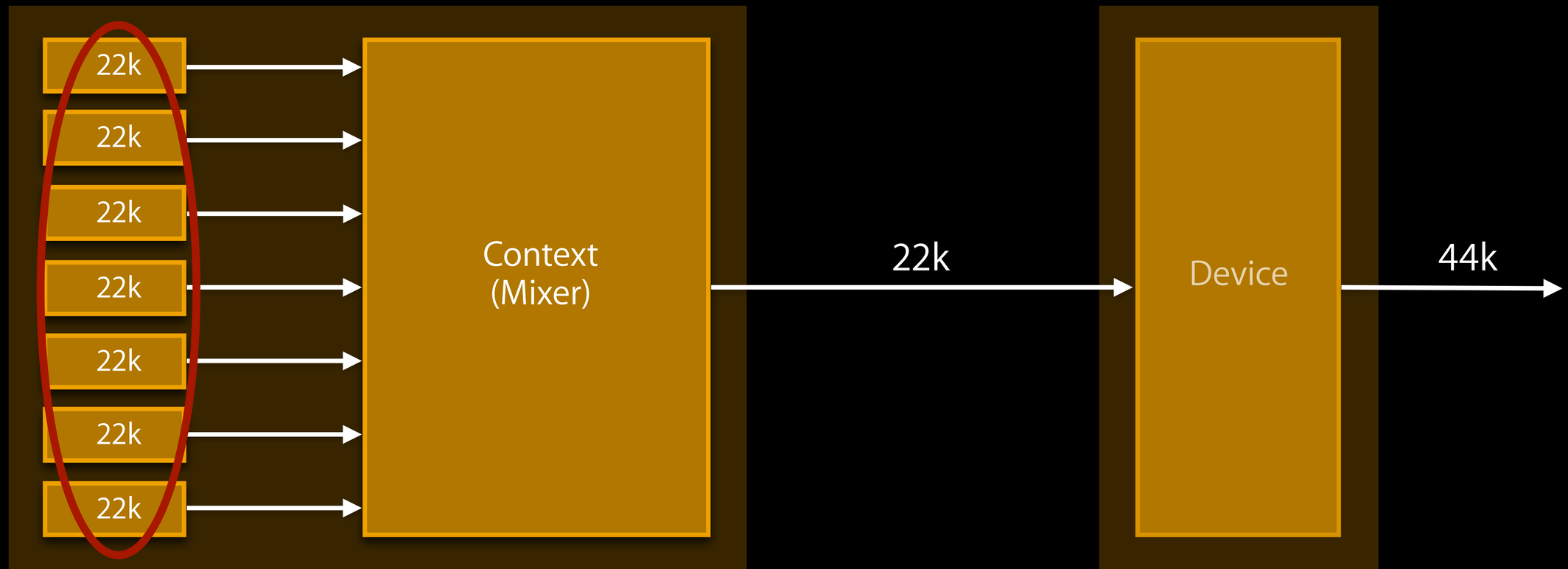
- Recommendation #1
  - Set mixer output rate to highest sample rate of source content
  - Call alcMixerOutputRate() before creating a context



OpenAL Best Practice

Tuning your assets

- Recommendation #2
  - Make source content the same sample rate as mixer output rate

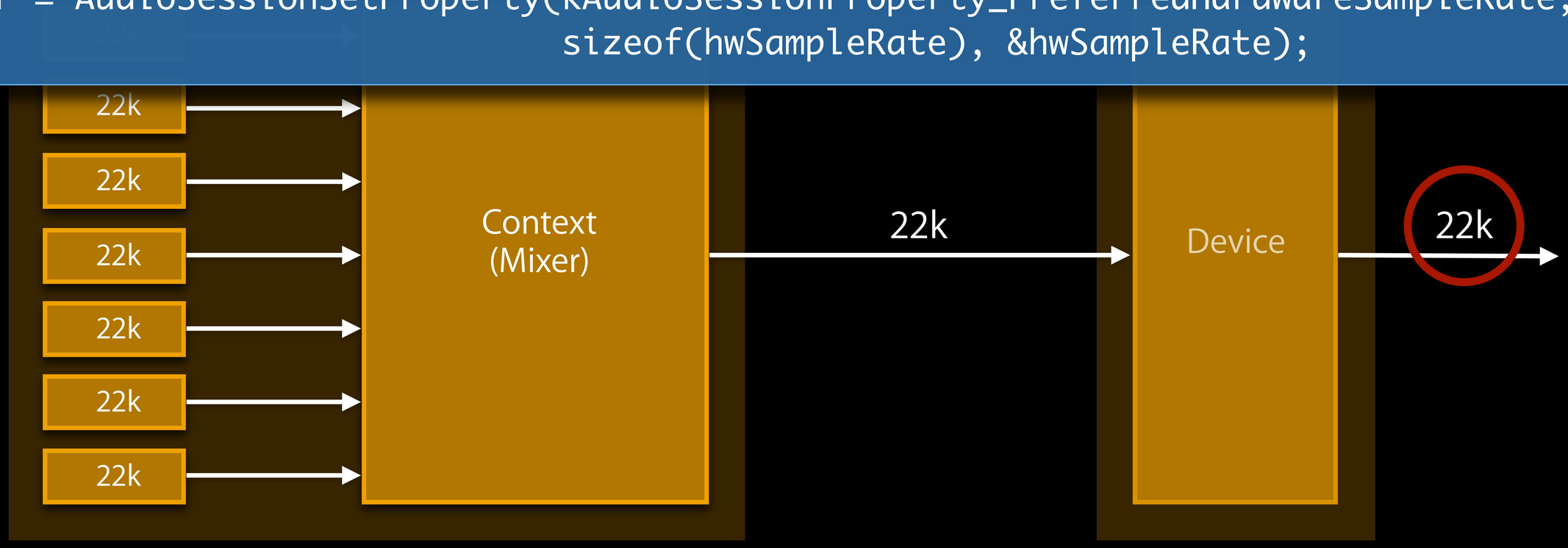


OpenAL Best Practice

Tuning your assets

- Recommendation #3
  - If not playing background audio track, consider setting HW to 22k

```
Float64   hwSampleRate = 22050.0;
err = AudioSessionSetProperty(kAudioSessionProperty_PreferredHardwareSampleRate,
                              sizeof(hwSampleRate), &hwSampleRate);
```



Input/Output Audio Units

Doug Wyatt
Audio/MIDI Plumber

Input/Output Audio Units

- Names
  - AUHAL (desktop)
  - AURemoteIO (iPhone)
- When to use one
- How they work
- How to use one

When to Use an I/O Audio Unit

- You need more reponsiveness, lower latency than provided with higher-level APIs
- You are prepared to execute in a constrained environment
- Examples
  - Custom game audio engine
  - IP telephony
  - Software instrument
- The Core Audio HAL is the lowest-level audio API but
  - Desktop: Complexities of multichannel devices
  - iPhone: HAL not available

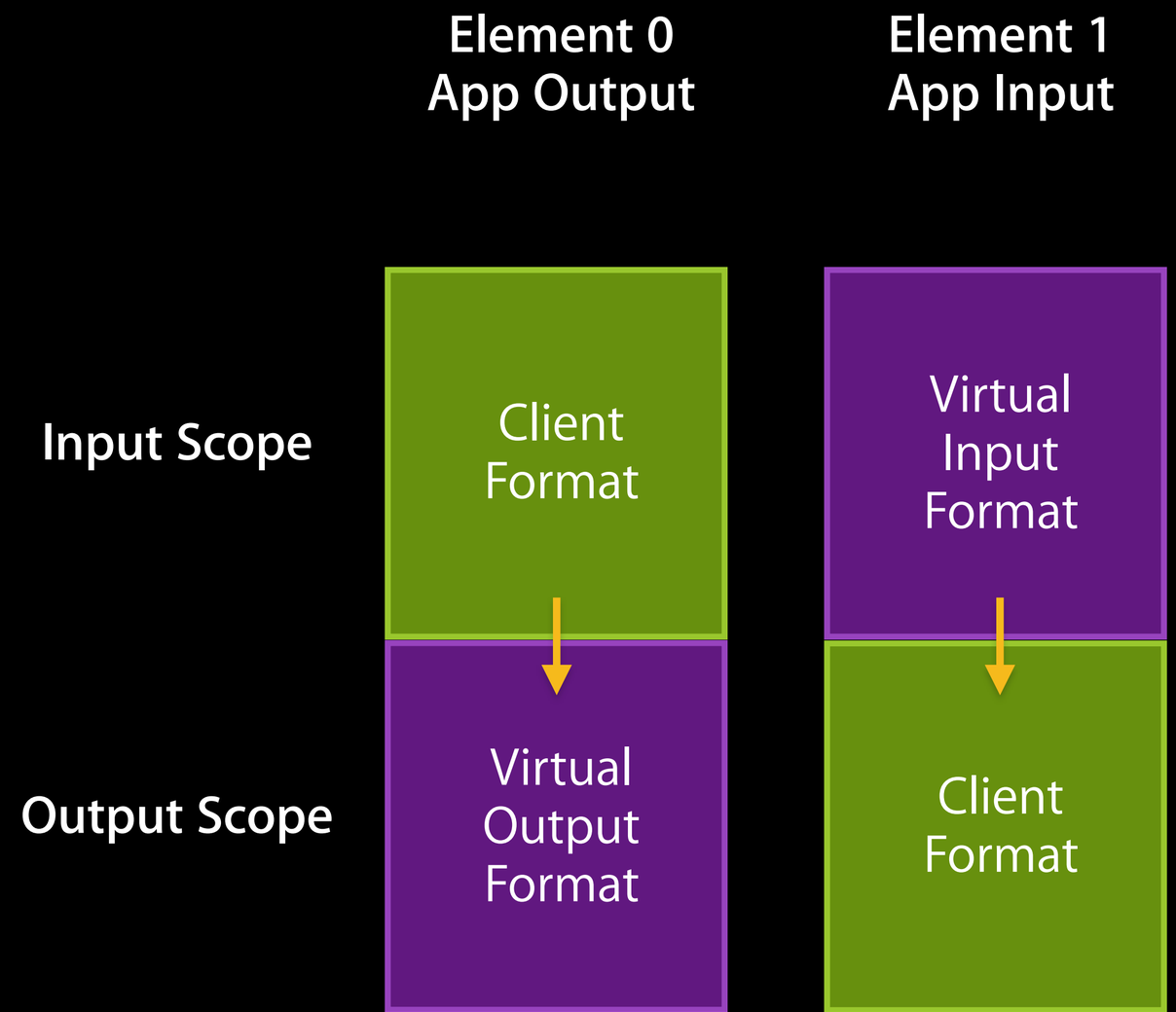
Overview of the I/O Audio Units

- You can configure the unit for input, output, or both
- You specify PCM “client formats”
- Output
  - From another Audio Unit (e.g. mixer, effect)
  - From your callback function
- Input
  - Your callback function is notified, calls AudioUnitRender
- AU converts to and from your client formats

Opening the Audio Unit

```
#include <AudioUnit/AudioUnit.h>

AudioUnit myIOUnit = NULL;
AudioComponentDescription desc;

desc.componentType = kAudioUnitType_Output;
desc.componentSubType = kAudioUnitSubType_RemoteIO;
desc.componentManufacturer = kAudioUnitManufacturer_Apple;
desc.componentFlags = 0;
desc.componentFlagsMask = 0;

AudioComponent comp = AudioComponentFindNext(NULL, &desc);

AudioComponentInstanceNew(comp, &myIOUnit);
```

Anatomy of an I/O Unit



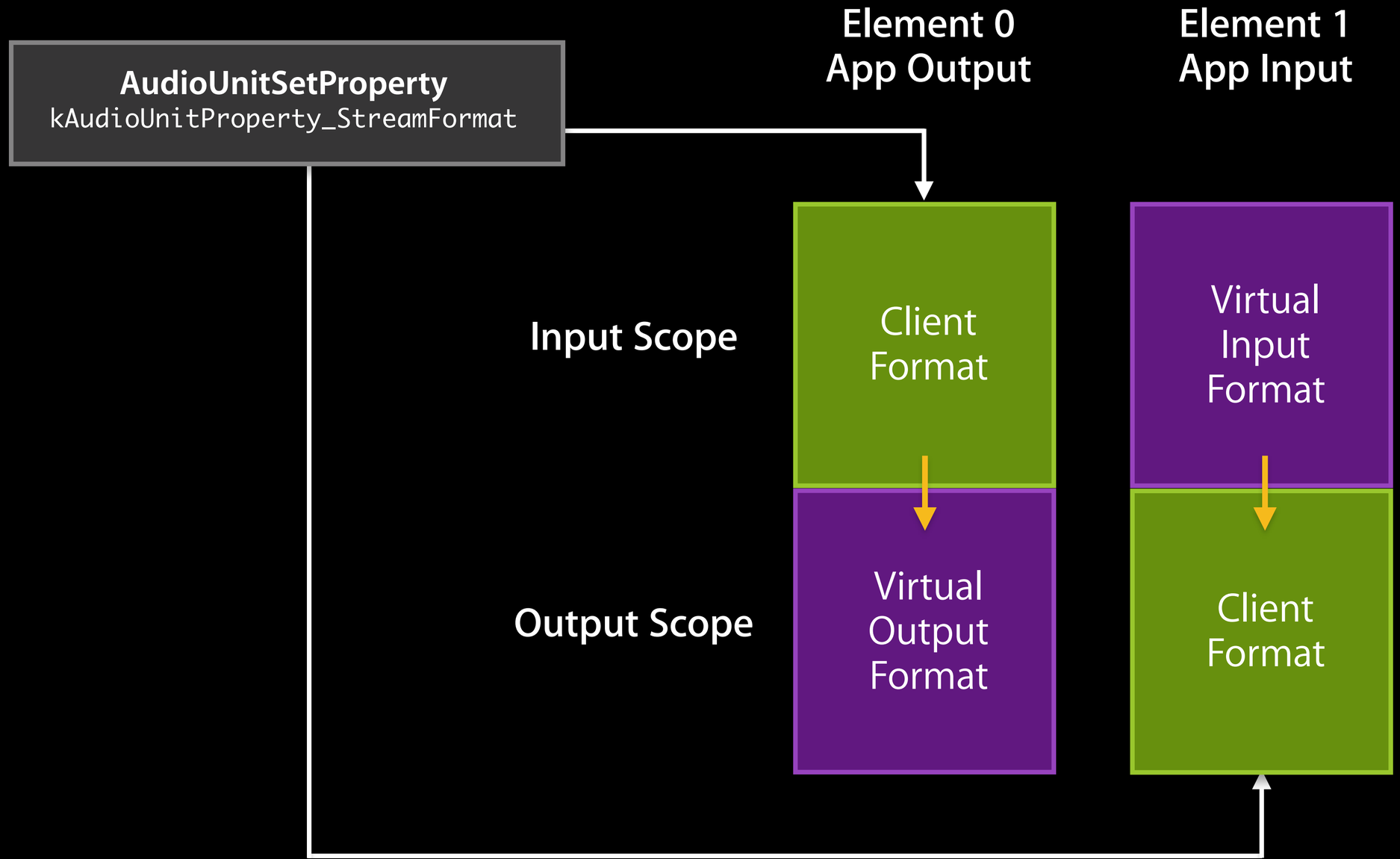
Considering Stream Formats—AURemoteIO

- If recording, ask AudioSession if input is available
- Use an AudioSession listener to observe rate changes
- Tell AudioSession your preferred sample rate
  - Note: Original iPhone will only record at 8kHz
- Call AudioUnitInitialize()
  - This may cause the sample rate to change to your session's preferred rate

Stream Formats—AURemoteIO

- Fetch the virtual I/O format(s)
- Choose your client format
  - AU can rate convert both input and output
  - Avoid rate conversions when possible
    - Do NOT simply copy the virtual I/O format!
    - Typically you might adjust your sample rate and channel count to match the virtual format

Setting Your Client Stream Formats



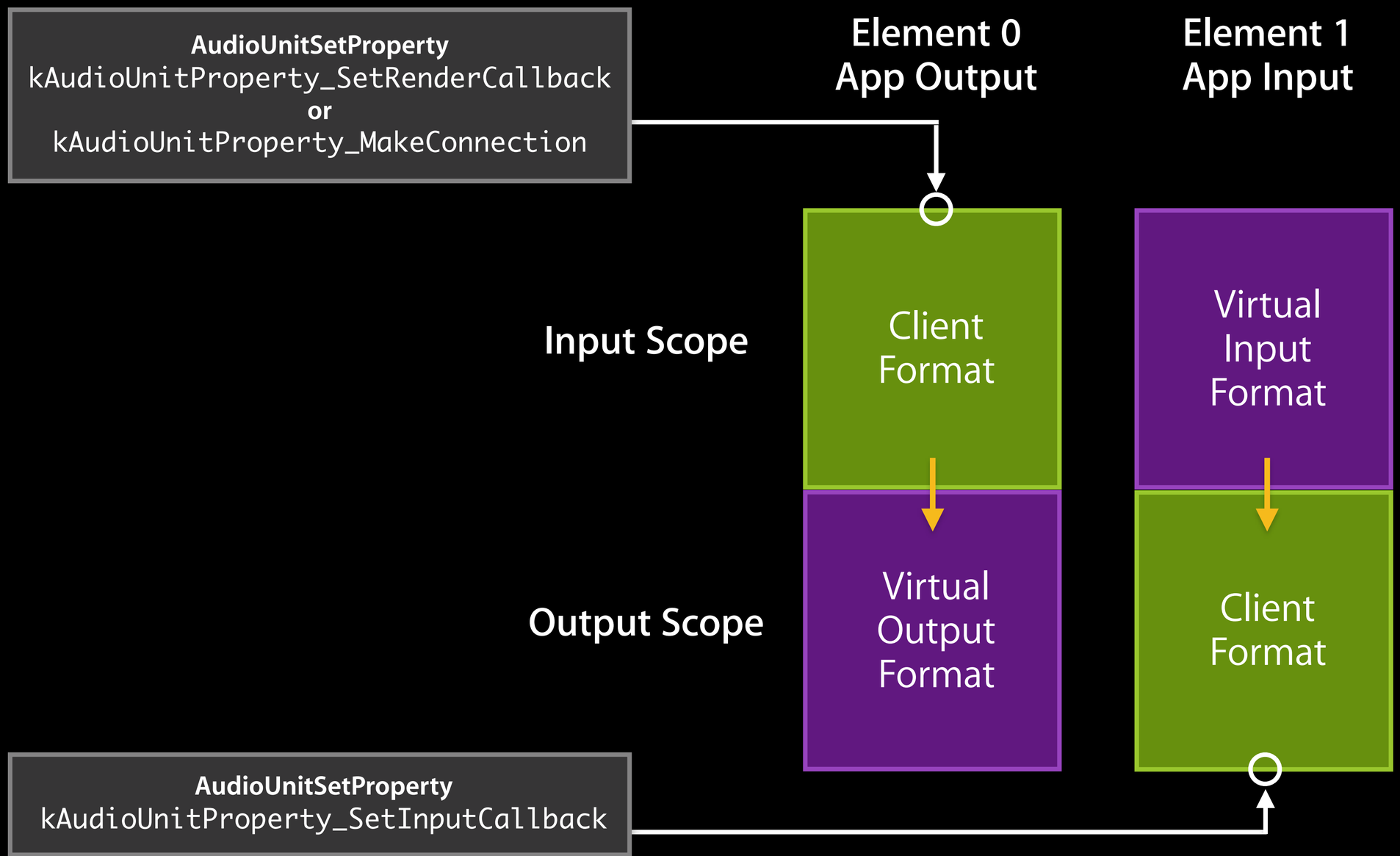
Stream Formats—AUHAL (Desktop)

- Fetch the hardware rate from the virtual I/O format
- Use AudioUnitAddPropertyListener() to observe rate changes
- Avoid changing the hardware sample rate
- Output
  - Client format may be at a different sample rate than the hardware; conversion is at a good quality
- Input
  - No sample rate conversion is provided by the AU
  - Must record at hardware rate
  - Can capture into a ring buffer and rate convert on a separate thread

Setting Your Client Stream Formats

```
AudioStreamBasicDescription asbd;
UInt32 propertySize = sizeof(asbd);

asbd.mSampleRate = 44100;
asbd.mFormatID = kAudioFormatLinearPCM;
/* etc... */
/* see sample code for examples of how to construct an ASBD */

AudioUnitSetProperty(myIOUnit,
  kAudioUnitProperty_StreamFormat,
  kAudioUnitScope_Input,
  0, /* element */
  sizeof(asbd), &asbd);
```

Connecting

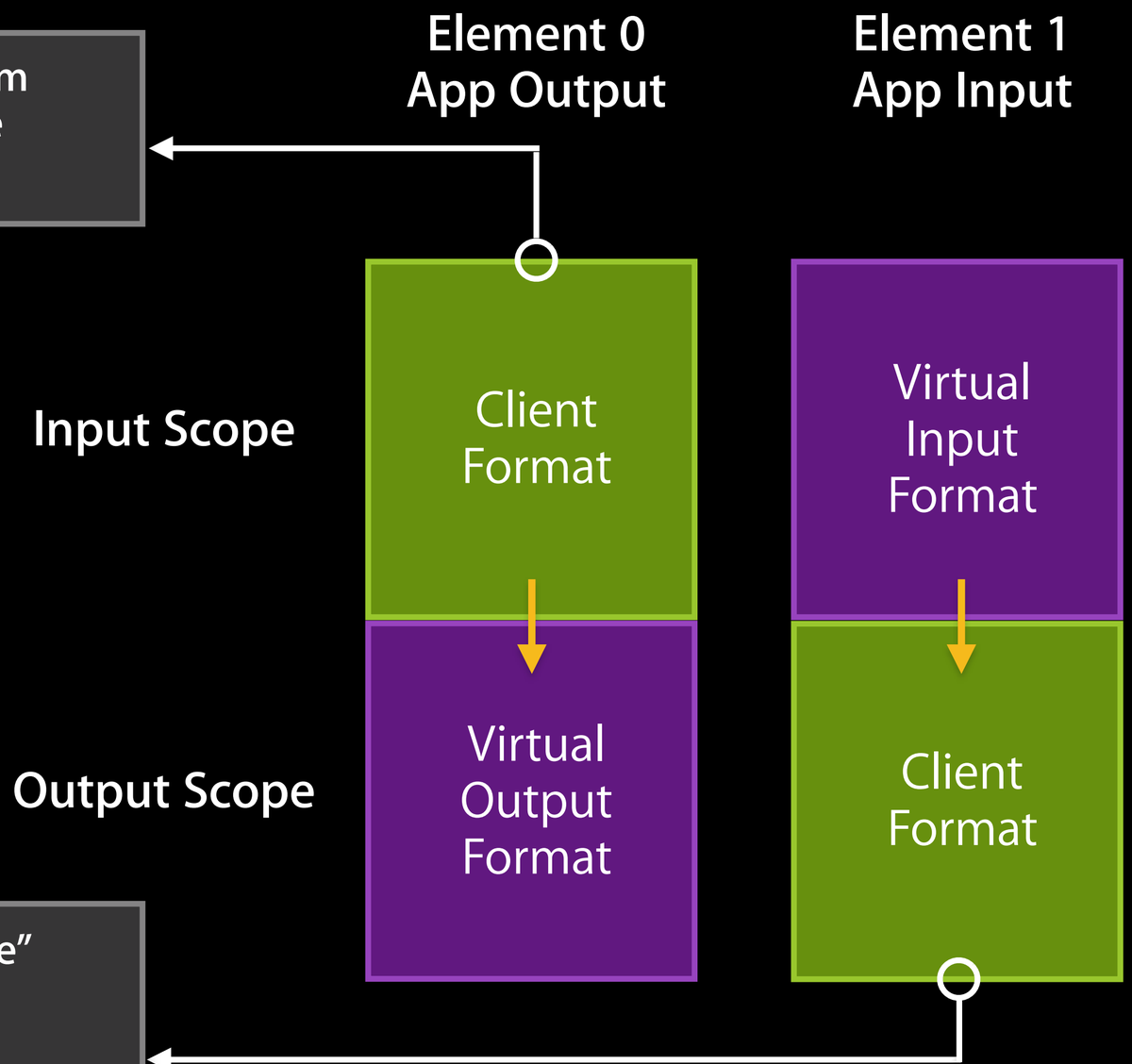

AudioUnitSetProperty
kAudioUnitProperty_SetRenderCallback
or
kAudioUnitProperty_MakeConnection
Element 0
App Output
Element 1
App Input
Input Scope
Output Scope
Client
Format
Virtual
Input
Format
Virtual
Output
Format
Client
Format
AudioUnitSetProperty
kAudioUnitProperty_SetInputCallback

Initializing and Starting

```
AudioUnitInitialize(myIOUnit);

AudioOutputUnitStart(myIOUnit);
```

Rendering

3. Your render callback or upstream audio unit is “pulled” to provide audio output



1. Input callback: “input is available”
2. You call AudioUnitRender to obtain the audio input

Audio I/O Callback Functions

- Callbacks run on a realtime priority thread
- Efficiency is important
- Do not
  - Allocate memory
  - Take a mutex (try/fail if you must, e.g. for reconfiguration)
  - Access the filesystem or network or use any other API that might block the thread
- Otherwise: You will glitch
- Avoid Objective C messaging
  - Do not invoke a method for every sample!

Example: Input Using AURemoteIO

```
#include <AudioUnit/AudioUnit.h>

// Find the AudioUnit
AudioUnit myIOUnit = NULL;
AudioComponentDescription desc;

desc.componentType = kAudioUnitType_Output;
desc.componentSubType = kAudioUnitSubType_RemoteIO;
desc.componentManufacturer = kAudioUnitManufacturer_Apple;
desc.componentFlags = 0;
desc.componentFlagsMask = 0;

AudioComponent comp = AudioComponentFindNext(NULL, &desc);

// Create an instance of the AudioUnit
AudioComponentInstanceNew(comp, &myIOUnit);
```

```
// Enable audio input
UInt32 one = 1;
AudioUnitSetProperty(myIOUnit, kAudioUnitProperty_EnableIO,
    kAudioUnitScope_Input, 1, &one, sizeof(one));

// Disable audio output
UInt32 zero = 0;
AudioUnitSetProperty(myIOUnit, kAudioUnitProperty_EnableIO,
    kAudioUnitScope_Output, 0, &zero, sizeof(zero));

// Initialize
AudioUnitInitialize(myIOUnit);

// Only after Initialize are the virtual I/O formats valid
```

```
// Examine input virtual I/O format
AudioStreamBasicDescription ioFormat, clientFormat;
UInt32 size = sizeof(ioFormat);
AudioUnitGetProperty(myIOUnit, kAudioUnitProperty_StreamFormat,
    kAudioUnitScope_Input, 1, &ioFormat, &size);

// Set input client I/O format
// e.g. we want 16-bit native-endian, stereo interleaved integers,
// at whatever the hardware sample rate is
clientFormat.mSampleRate = ioFormat.mSampleRate;
clientFormat.mFormatID = kAudioFormatLinearPCM;
clientFormat.mFormatFlags = kAudioFormatFlagIsSignedInteger
                  | kAudioFormatFlagsNativeEndian
                  | kAudioFormatFlagIsPacked;
clientFormat.mBitsPerChannel = 16;
clientFormat.mChannelsPerFrame = 2;
clientFormat.mFramesPerPacket = 1;
clientFormat.mBytesPerPacket = clientFormat.mBytesPerFrame =
    clientFormat.mChannelsPerFrame * sizeof(SInt16);

AudioUnitSetProperty(myIOUnit, kAudioUnitProperty_StreamFormat,
    kAudioUnitScope_Output, 1,
    &clientFormat, sizeof(clientFormat));
```

```
// Prepare variables to be used in our input callback
struct MyContext {
    AudioUnit                   myIOUnit;
    AudioBufferList *           myInputBufferList;
    AudioStreamBasicDescription  myInputFormat;
} myContext;

myContext.myIOUnit = myIOUnit;
myContext.myInputBufferList = (AudioBufferList *)
    malloc(sizeof(AudioBufferList));
myContext.myInputFormat = clientFormat;
```

```
// Specify an input callback
AURenderCallbackStruct rc;
rc.inputProc = MyRecordInput;
rc.inputProcRefCon = (void *)&myContext;
AudioUnitSetProperty(myIOUnit,
    kAudioUnitProperty_SetInputCallback,
    kAudioUnitScope_Global, 0, &ioFormat, &size);

// Start performing I/O
AudioOutputUnitStart(myIOUnit);
```

```
// Input callback -- called when a buffer of input is available
OSStatus MyRecordInput(void * inRefCon,
    AudioUnitRenderActionFlags *     ioActionFlags,
    const AudioTimeStamp *           inTimeStamp,
    UInt32                           inBusNumber,
    UInt32                           inNumberFrames,
    AudioBufferList *                ioData)
{
    myContext *ctxt = (myContext *)inRefCon;
    // Prepare our AudioBufferList, into which input will be converted.
    AudioBufferList *abl = ctxt->mInputBufferList;
    // This code knows that we specified a stereo interleaved format earlier.
    abl->mNumberBuffers = 1;
    abl->mBuffers[0].mNumberChannels = 2;
    abl->mBuffers[0].mData = NULL;
    abl->mBuffers[0].mDataByteSize = inNumberFrames *
        ctxt->myInputFormat.mBytesPerFrame;
    AudioUnitRender(ctxt->myIOUnit, ioActionFlags, inTimeStamp, 1,
        inNumberFrames, abl);

    // ... your code here to process input in ctxt->myInputBufferList ...

    return noErr;
}
```

Summary

- Use AURemoteIO and AUHAL for low-latency audio I/O
- Always specify your client data formats; assume nothing
- In I/O callbacks, avoid memory allocation, semaphores/mutexes, and Objective-C method dispatching

Voice Processing Audio Unit

Eric Allamanche
Core Audio Engineering

Outline

- Why a Voice Processing Audio Unit?
- Functionality of an Acoustic Echo Canceler
- Features
- Using the Voice Processing Audio Unit in your code
- Demo

Why a Voice Processing Unit?

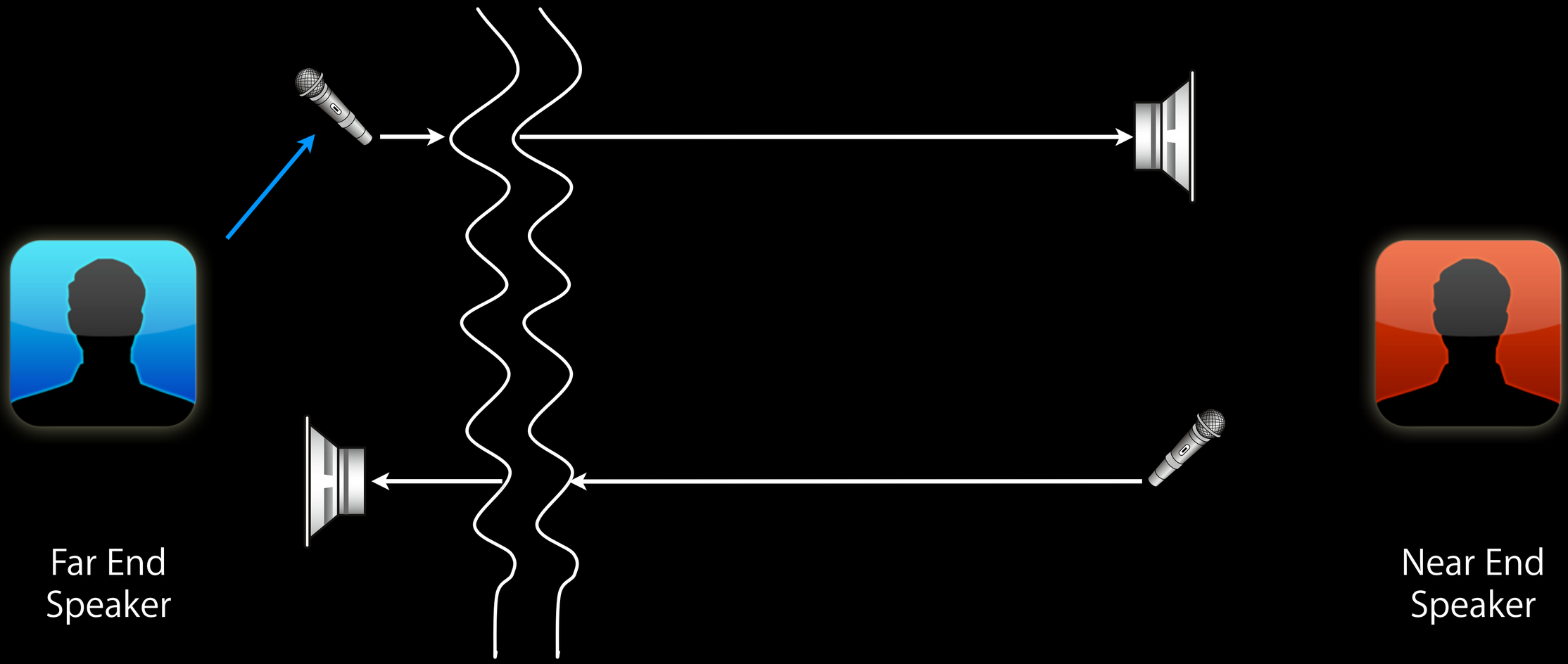
- In full-duplex audio systems, loudspeaker and microphone are in use simultaneously
- Microphone captures the sound emitted by the loudspeaker
  - In a telephony like application this sound is looped back to the sender
    - Due to the various delays in the processing and transmission chain (~ 100ms), this signal is perceived as an echo
- General problem in telephony (analog and digital) and voice-chat applications
- Solution: Apply an Acoustic Echo Canceler (AEC) to the captured signal

Why a Voice Processing Unit?

Why a Voice Processing Unit?

- The telephony part of the iPhone has its own, built-in, echo canceler
- Adding a new Audio Unit in iPhone OS 3.0 which you can use in your application

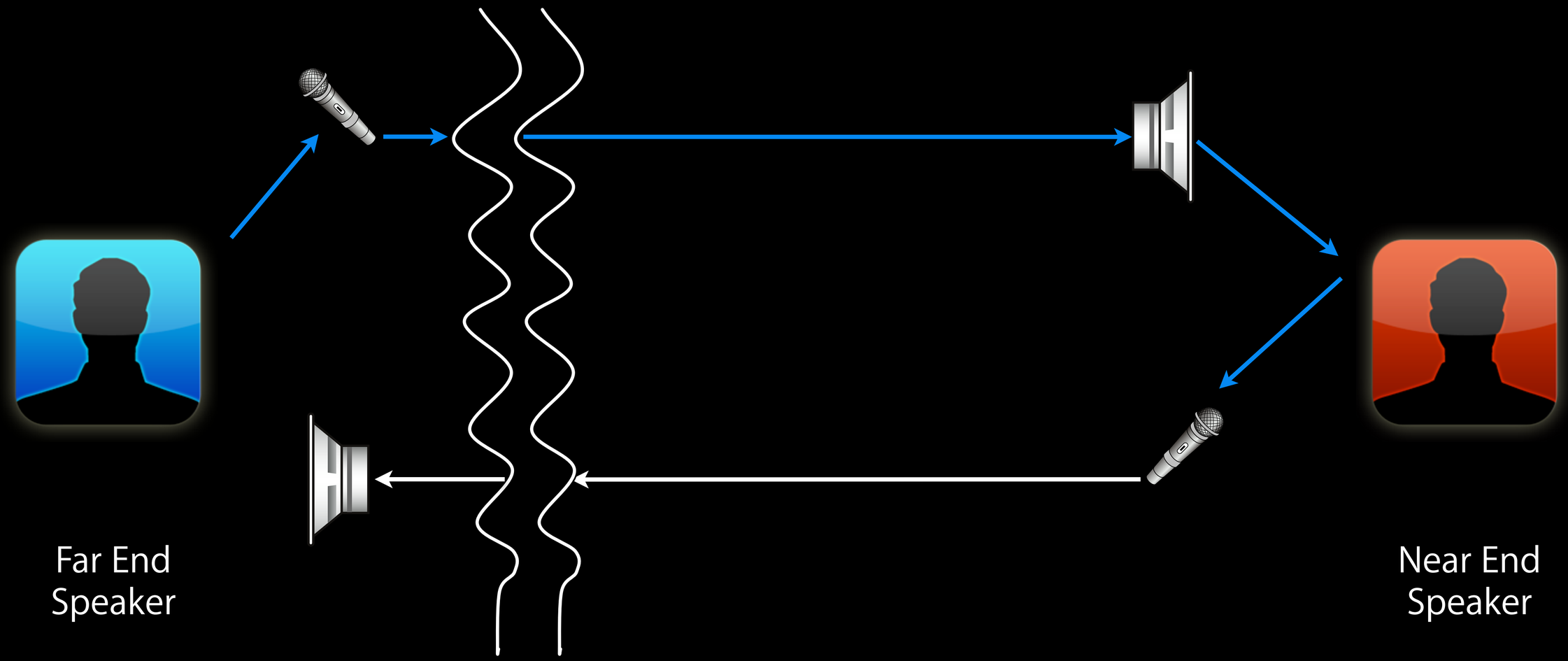
Functionality of an Acoustic Echo Canceler



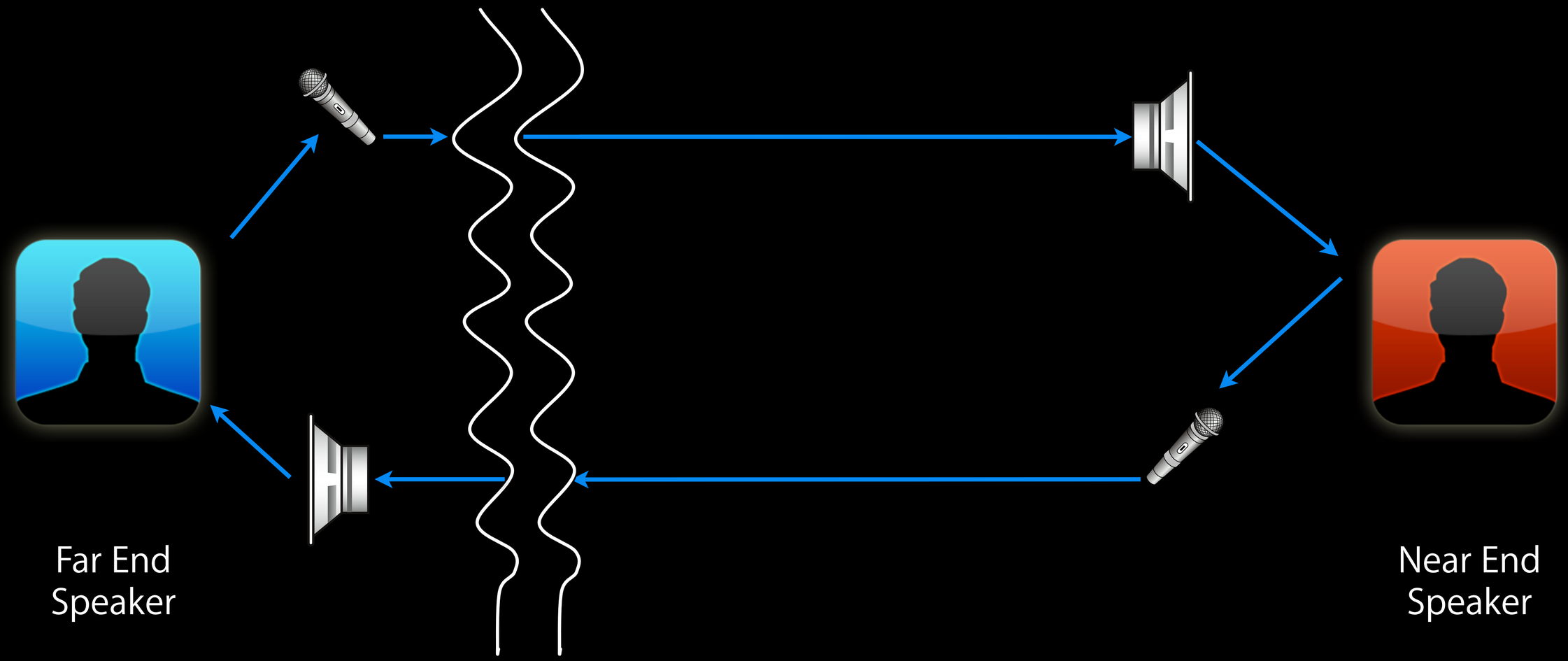
Far End
Speaker

Near End
Speaker

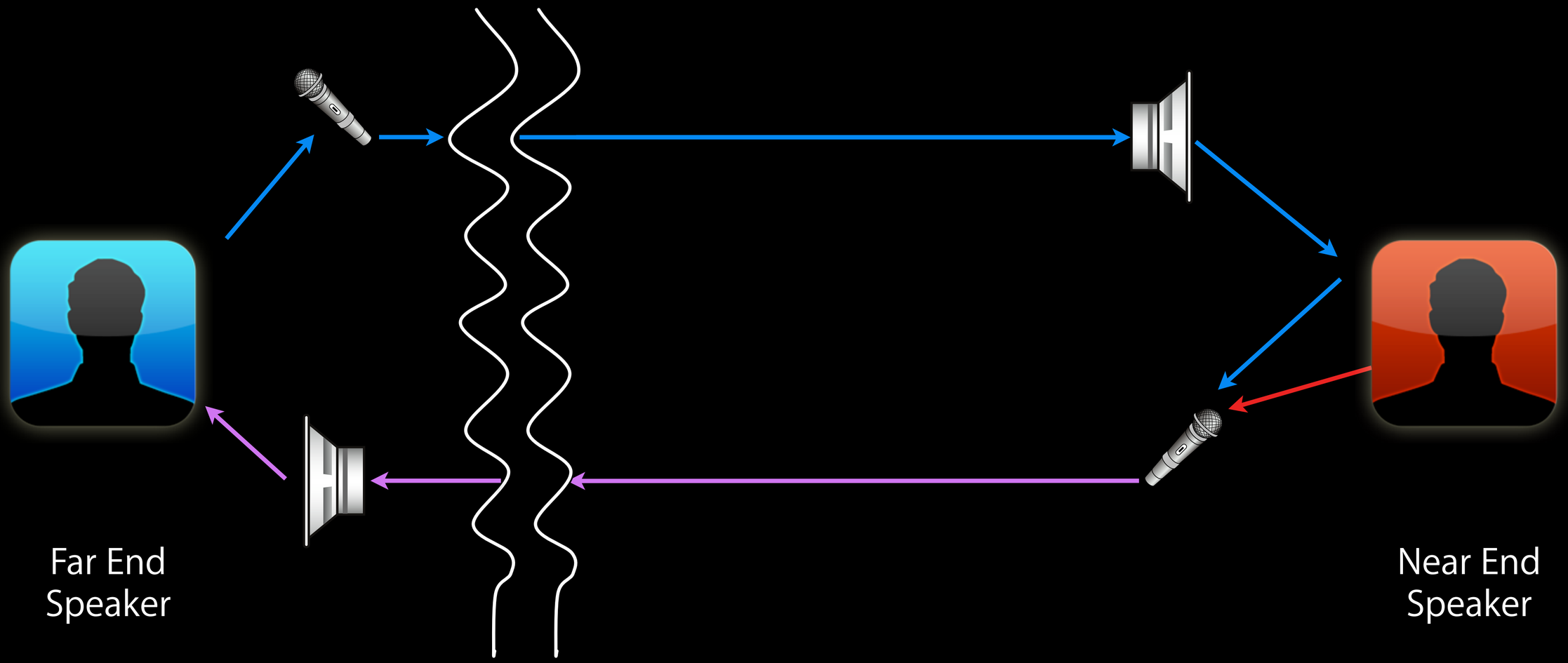
Functionality of an Acoustic Echo Canceler



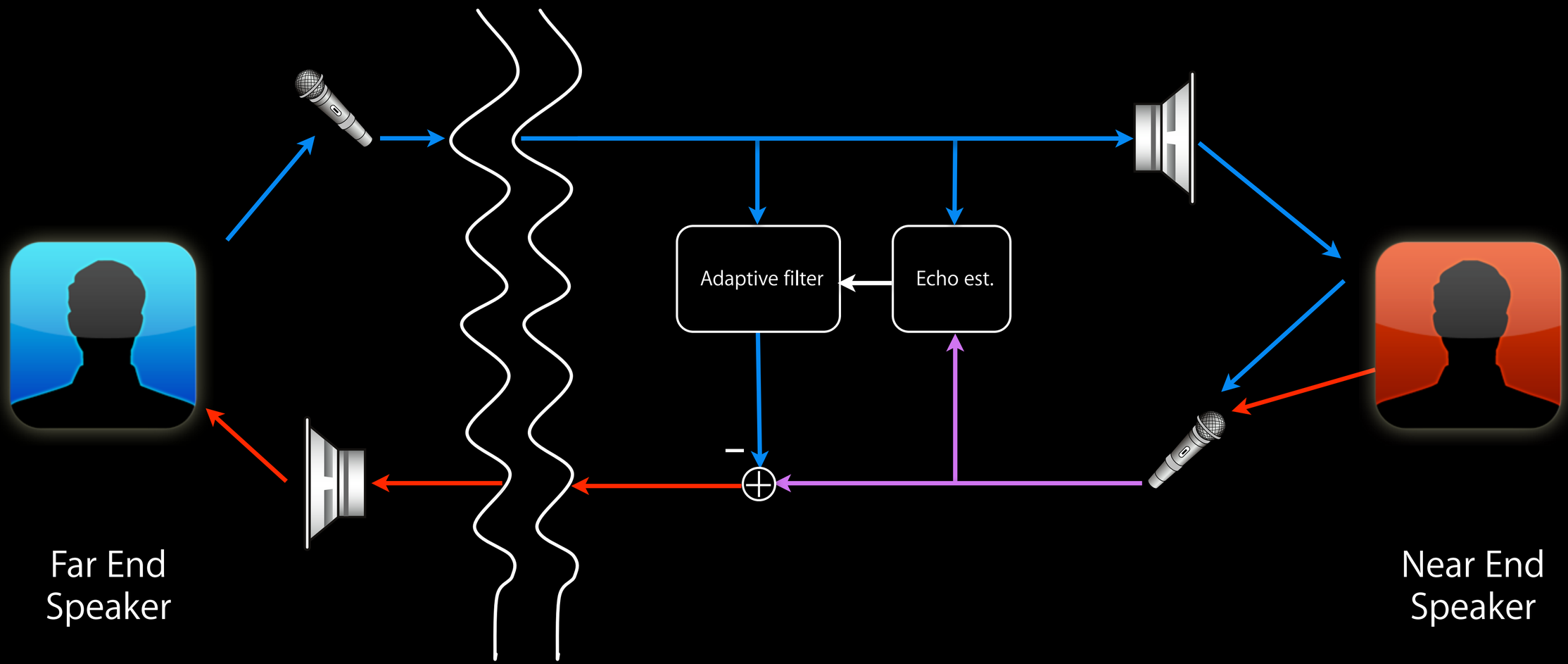
Functionality of an Acoustic Echo Canceler



Functionality of an Acoustic Echo Canceler



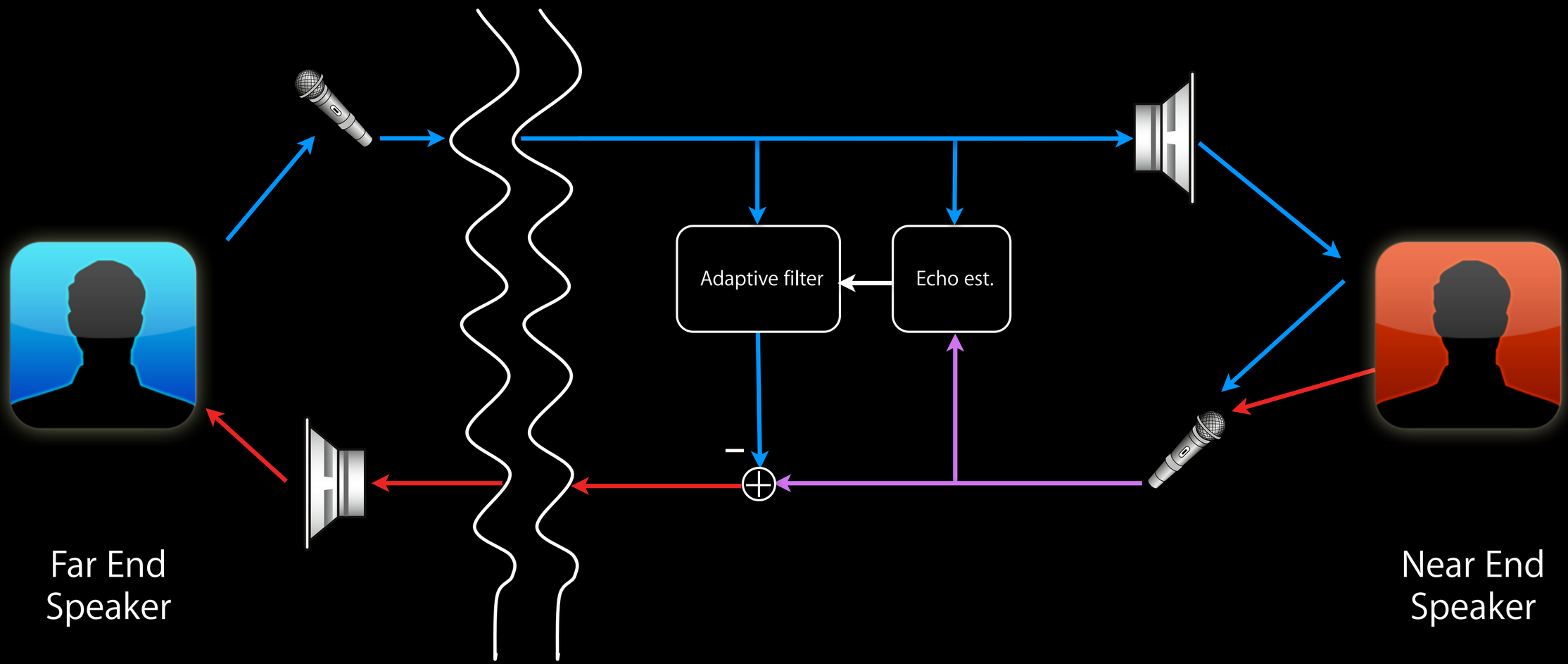
Functionality of an Acoustic Echo Canceler



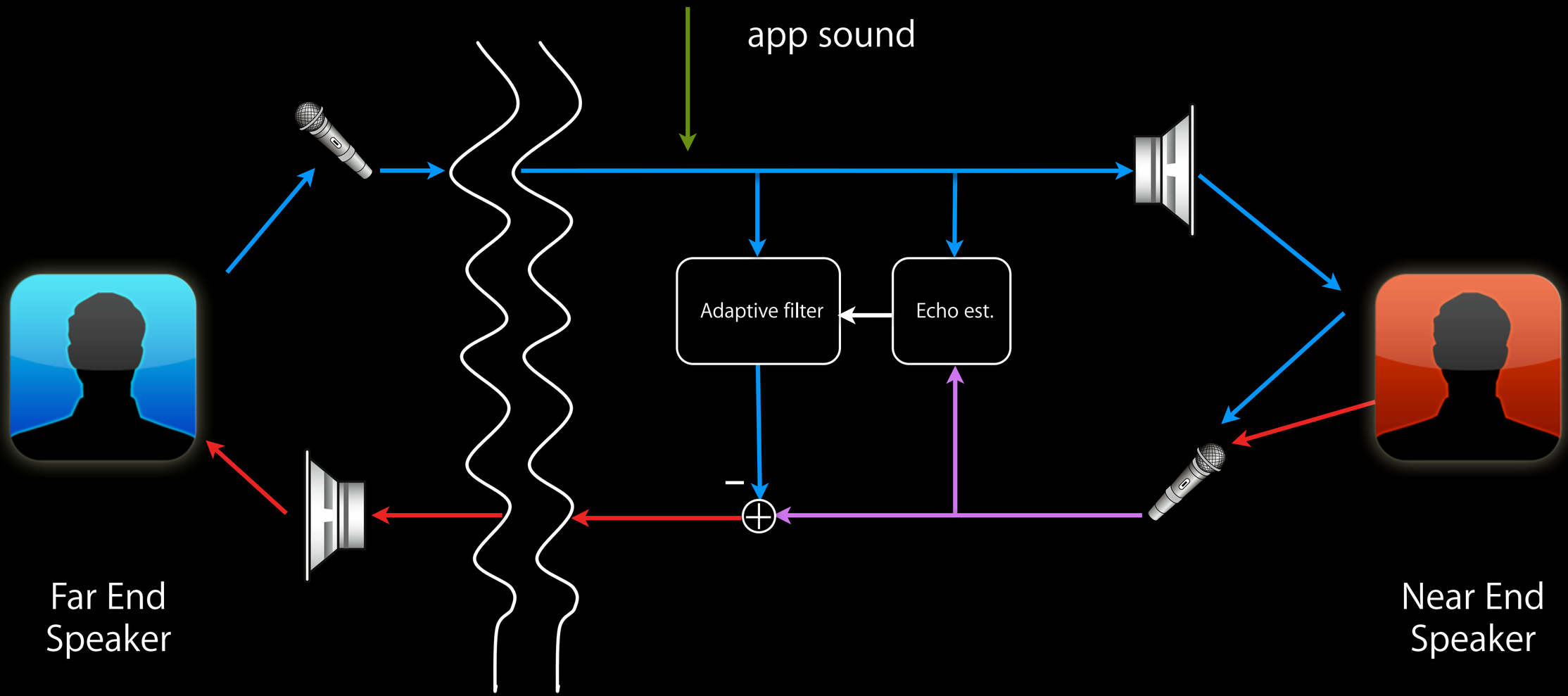
What Happens When a Sound Is Played?

- All signals and the hardware are at the same sample rate
- When playing additional sounds, these will be mixed into the rendering path and played back over the speaker
  - May have a lower bandwidth due to a down sampling
  - Can be heard by the far end speaker if that sound is mixed after the AEC

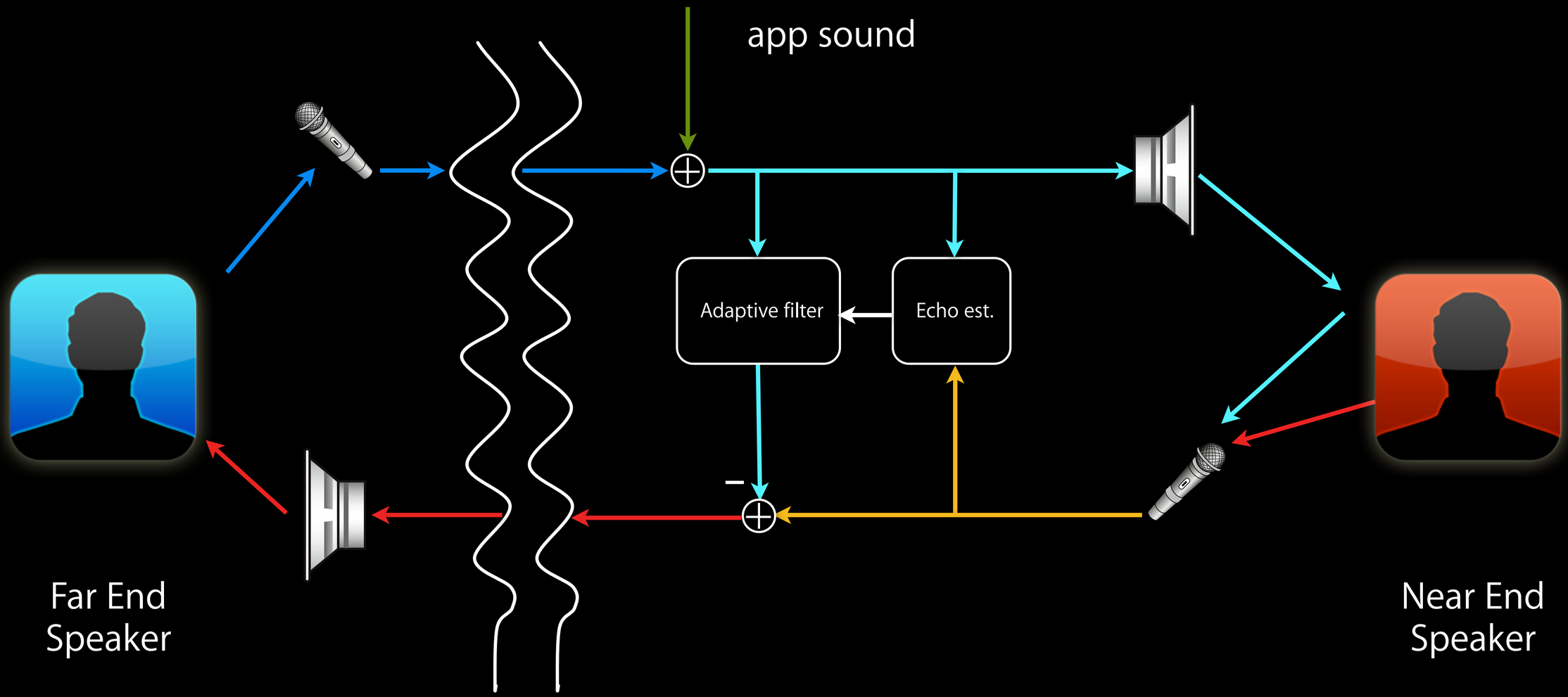
Functionality of an Acoustic Echo Canceler



Functionality of an Acoustic Echo Canceler



Functionality of an Acoustic Echo Canceler



Features

- Two operation modes
  - Single sample rate: The audio device and the voice client have the same sample rate
    - Voice chat only app
  - Dual sample rate: The audio device and the voice client have different sample rates—This allows to playback UI sounds or other sound effects with a high quality (app sound)
    - Chat while playing a game

Features

- UI and other app sounds are also cancelled since they are mixed inside the unit
- Optional automatic gain control (AGC)
- Supported voice client sample rates:
  - 8kHz, 11.025kHz, 12kHz, and 16kHz
- Secondary audio signal is ducked to maintain intelligibility
- But
  - Audio unit does not handle encoding or decoding (formatID is ‘lpcm’)

Properties

- Defined in the <AudioUnit/AudioUnitProperties.h> header file

`kAUVoiceIOProperty_BypassVoiceProcessing`

`kAUVoiceIOProperty_VoiceProcessingEnableAGC` (on by default)

`kAUVoiceIOProperty_DuckNonVoiceAudio` (on by default)

Setting Up the Voice Processing Unit

- Setup a `kAudioSessionCategory_PlayAndRecord` audio session to enable simultaneous audio I/O
  - Set the preferred hardware sample rate
    - This will enable the dual rate mode if this sample rate differs from the voice client one
- Create a `kAudioUnitSubType_VoiceProcessingIO` Audio Unit, this is similar to setting up a kAudioUnitSubType_RemoteIO unit
- Implement both InputProc and RenderProc callbacks

Demo

Summary

- AVAudioPlayer/Recorder
  - Simplest way to playback and record audio in all formats supported by the device
- OpenAL
  - Best practices
- AURemoteIO audio unit
  - Close to hardware, low latency
- Voice Processing IO audio unit
  - Same interface as AURemoteIO, adds multi-rate echo canceler

Related Sessions

Audio Development for iPhone OS	Presidio Tuesday 3:30PM
Peer to Peer Networking with Game Kit	Presidio Wednesday 9:00AM

Labs

Audio Lab	iPhone Lab B Wednesday 12:00PM
Audio Lab	Graphics Lab A Wednesday 3:30PM

More Information

Allan Schaffer
Graphics Technology Evangelist
aschaffer@apple.com

Q&A

WWDC Session Survey

You can rate your sessions on the WWDC Attendee Site.

Survey forms are located on each session description page at:

developer.apple.com/wwdc/attendee

WWDC09